Fundamentos de Algoritmia

Grados en Ingeniería Informática. Grupos A, E y DG

Examen convocatoria extraordinaria, 5 de julio de 2021.

Normas de realización del examen

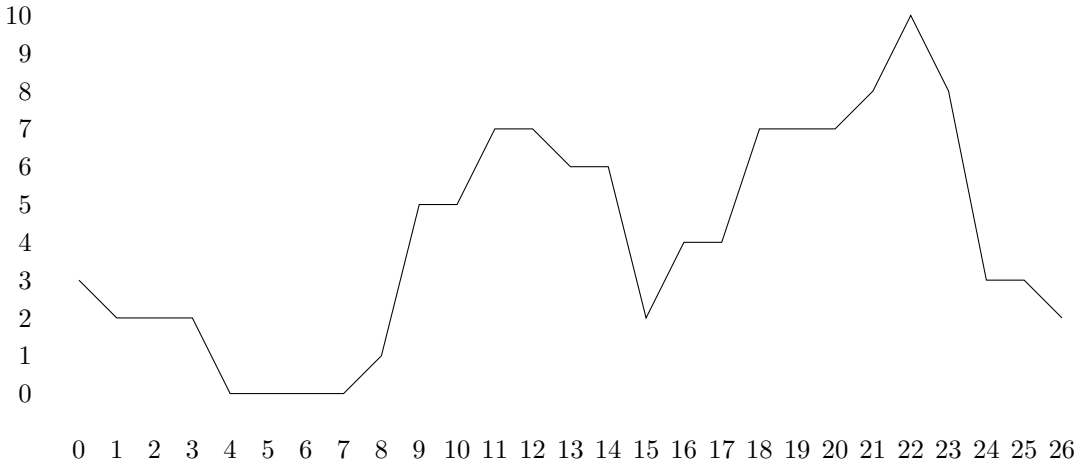
1. Debes programar soluciones para cada uno de los tres ejercicios, probarlas y entregarlas en el juez automático accesible en la dirección `http://exacrc`.
2. Escribe comentarios que expliquen tu solución, justifiquen por qué se ha hecho así y ayuden a entenderla. Calcula la complejidad de todas las funciones que implementes.
3. En el juez te identificarás con el nombre de usuario y contraseña que has recibido al comienzo del examen.
4. Escribe tu **nombre y apellidos** en un comentario en la primera línea de cada fichero que subas al juez.
5. Tus soluciones serán evaluadas por la profesora independientemente del veredicto del juez automático. Para ello, la profesora tendrá en cuenta **exclusivamente** el último envío que hayas realizado de cada ejercicio.

1. Travesía por los Alpes

(3.5 puntos) Kilian Korretonet está preparando su gran travesía por los Alpes. Ha representado su plan de travesía mediante una secuencia de valores naturales que indican las alturas de los puntos por los que tiene que pasar. De esta forma, la travesía está formada por *fases ascendentes* (secuencia no vacía de valores no decrecientes) alternando con *fases descendentes* (secuencia no vacía de valores no crecientes).

En una fase ascendente se pueden distinguir varios *tramos* (secuencia estrictamente creciente con al menos dos valores y maximal) separados por *llanos* (secuencia de valores iguales con al menos dos valores y maximal).

Por ejemplo la secuencia: `[3,2,2,2,0,0,0,0,1,5,5,7,7,6,6,2,4,4,7,7,7,8,10,8,3,3,2]` representa una travesía con una fase descendente de longitud 8 que comprende los valores del primer `3` hasta el cuarto `0` inclusive, seguido de una fase ascendente de longitud 9 que comprende los valores del primer `0` hasta el último `7`, ambos inclusive. Sigue una fase descendente de longitud 5, otra fase ascendente de longitud 8, y una última fase descendente de longitud 5. Por lo tanto, la fase ascendente más larga tiene longitud 9 y tiene dos tramos (y tres llanos).



Dado un vector v no vacío de enteros no negativos:

a) Define un predicado $ascenso(v, p, q)$ que devuelva cierto si y solo si el segmento $v[p..q]$ es una fase ascendente.

b) Define un predicado $tramo(v, p, q)$ que devuelva cierto si y solo si el segmento $v[p..q]$ es un tramo (recuerda que los tramos son maximales).

c) Utilizando los predicados anteriores, especifica una función que dado un vector no vacío de enteros no negativos devuelva la longitud de la fase ascendente más larga, el comienzo y el final de una fase ascendente de longitud máxima, y el número de tramos en dicha fase.

d) *Diseña e implementa* un algoritmo iterativo que resuelva el problema especificado. Si hubiera varias fases ascendentes con la misma longitud, los valores se referirán a la fase más a la izquierda.

e) Escribe el *invariante* del bucle que permite demostrar la corrección del mismo y proporciona una *función de cota*.

f) Indica el *coste asintótico* del algoritmo en el caso peor y justifica adecuadamente tu respuesta.

Entrada

La entrada comienza con una línea que contiene el número de casos de prueba. Cada caso de prueba estará formado por dos líneas: la primera contendrá el número de valores y la segunda línea contendrá los valores de la secuencia.

Salida

Por cada caso de prueba el programa escribirá una línea con la longitud de la fase ascendente más larga, el comienzo y el final de la fase y el número de tramos en dicha fase.

Entrada de ejemplo

```
7
27
3 2 2 2 0 0 0 0 1 5 5 7 7 6 6 2 4 4 7 7 7 8 10 8 3 3 2
1
3
3
3 2 1
3
1 2 3
4
2 2 3 3
4
2 2 2 2
8
1 2 3 3 2 3 3 4
```

Salida de ejemplo

```
9 4 12 2
1 0 0 0
1 0 0 0
3 0 2 1
4 0 3 1
4 0 3 0
4 0 3 1
```

2. Travesía por el desierto

(2.5 puntos) En cualquier travesía por el desierto un buen suministro de agua es fundamental para poder llegar al destino. La compañía Solyarena, especialista en viajes en grupo a lugares desérticos ha desarrollado una aplicación que le permite detectar cuándo debe racionar el agua que proporciona a los excursionistas. Para ello cuenta con una estimación de la cantidad de agua disponible en el depósito al comenzar cada día de travesía. El objetivo es encontrar el primer día en el que debe aplicarse un racionamiento del agua porque la cantidad de agua estimada es insuficiente para dar a cada miembro del grupo un litro por cada día que resta de viaje.

a) Escribe un algoritmo recursivo *eficiente* que permita resolver el problema.

b) Escribe la *recurrencia* que corresponde al coste de la función recursiva indicando claramente cuál es el *tamaño del problema*. Indica también a qué *orden de complejidad asintótica* pertenece dicho coste.

Entrada

La entrada comienza con una línea que contiene el número de casos de prueba. Cada caso de prueba contendrá inicialmente el valor del número de días que el grupo permanecerá en el desierto y el número de integrantes del grupo (estos dos valores son mayores que cero). En la línea siguiente se tiene la estimación de los litros de agua que quedarán cada día en el depósito como una secuencia de valores enteros que decrecen al menos tanto como el número de integrantes del grupo.

Salida

Para cada caso de prueba el programa escribirá el día en que debe comenzar el racionamiento (el primer día es el día cero) o `SIN RACIONAMIENTO` en caso de que no haya que aplicarlo.

Entrada de ejemplo

```
4
7 1
20 15 10 5 2 1 0
6 2
20 2 0 -3 -6 -10
5 3
40 35 30 25 20
6 4
20 16 12 8 4 0
```

Salida de ejemplo

```
4
1
SIN RACIONAMIENTO
0
```

3. Travesía en canoa.

(4.0 puntos) Un grupo de M amigos se ha propuesto descender en canoa el río Cares/Deva. La agencia de alquiler dispone de N canoas, todas ellas de las mismas características. Cada canoa puede transportar una, dos o tres personas, siempre y cuando no se superen k kilos. Se conocen los pesos (en kilos) de cada miembro del grupo de amigos. Para ahorrar costes, se quiere alquilar el número mínimo de canoas, pero además del peso limitado hay otro problema: las rencillas establecidas entre los amigos. Dos amigos enemistados (las enemistades son mutuas) no pueden compartir canoa.

a) Define el *espacio de soluciones* e indica cómo es el *árbol de exploración*.
b) Implementa un algoritmo de *vuelta atrás* que resuelva el problema. Explica claramente los *marcadores* que has utilizado.
c) Plantea al menos una *función de poda de optimalidad* e impleméntala en tu algoritmo.

Entrada

La entrada comienza con una línea que contiene el número de casos de prueba. Cada caso de prueba contendrá inicialmente el valor del número de amigos M, de canoas N y del peso máximo que soporta una canoa k. A continuación una fila con los M pesos de los amigos. Y finalmente M filas que indican mediante 0s (falso) y 1s (cierto) si dos amigos están o no enemistados.

Salida

Por cada caso de prueba el programa escribirá la cantidad mínima de canoas necesarias para realizar el descenso del río, respetando las restricciones indicadas. Se escribirá `IMPOSIBLE` si no se puede establecer un reparto de canoas adecuado.

Entrada de ejemplo

```
3
5 4 100
50 20 50 10 60
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
5 4 100
50 20 50 10 60
0 1 1 1 0
1 0 0 0 0
1 0 0 1 1
1 0 1 0 0
0 0 1 0 0
5 4 100
50 20 50 10 60
0 1 1 1 1
1 0 1 1 1
1 1 0 1 1
1 1 1 0 1
1 1 1 1 0
```

Salida de ejemplo

```
2
3
IMPOSIBLE
```